PES UNIVERSITY

(Established under Karnataka Act No. 16 of 2013)

100-ft Ring Road, Bengaluru – 560 085, Karnataka, India

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGG

PROGRAM B.TECH

Machine Learning(UE20EC352)

Project Report on

Vehicular Networking

And

Delay Estimation

Submitted By

– B Gautham PES1UG20EC044

– Sai Pranay Chennamsetti PES1UG20EC048

– Dheemanth R Joshi   PES1UG20EC059

1 Introduction and Motivation

The chosen domain for the final project was *Networking and Delay Estimation*. Networking refers to the process of connecting multiple devices or systems together to share resources and communicate with each other. The goal of networking is to establish a seamless and efficient communication system between different devices or systems, regardless of their physical location. Networks can be of different types, such as vehicular networks, transportation networks, or communication networks, and can include various devices or components, such as routers, switches, servers, sensors, and client devices. Effective network management is essential to ensure that the network operates smoothly, and network delays or congestion are minimized, allowing for faster and more efficient communication. Machine learning techniques can be used to improve network management and delay estimation, providing network administrators with a more effective means of managing and optimizing network performance.

(a) Vehicular Traffic

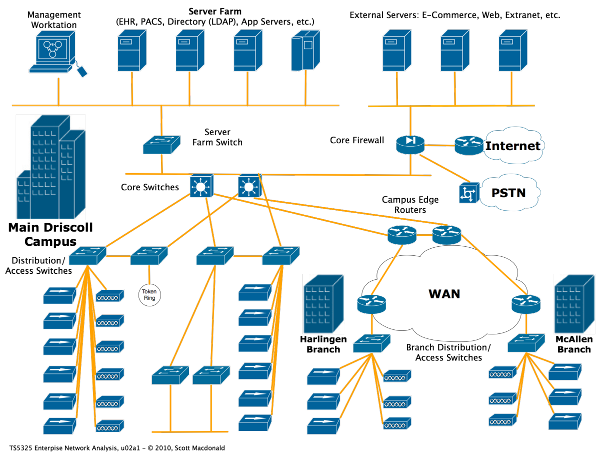

Management Worktation
Server Farm
(EHR, PACS, Directory (LDAP), App Servers, etc.)
External Servers: E-Commerce, Web, Extranet, etc.
Server Farm Switch
Core Firewall
Internet
PSTN
Core Switches
Campus Edge Routers
Main Driscoll Campus
Distribution/ Access Switches
Token Ring
WAN
Harlingen Branch
Branch Distribution/ Access Switches
McAllen Branch
TS5325 Enterpise Network Analysis, u02a1 – © 2010, Scott Macdonald
(b) Internet Traffic

Figure 1: Networking in different fields

In this report we will be addressing the problem of Network delays, in particular we will address the problem of *forecasting* the delays incurred in a network. This report contains the results for three datasets. Two of the datasets deal with the problem of estimating the vehicular traffic flowing through a junction and the third dataset deals with the processing delay encountered for a task offloaded to a server.

One of the key goals of 5G is to onboard large amount of IoT devices and users, by using the current infrastructure all the data needs to be sent to a remote central location (Cloud) but this increases the network delay (Latency), to recude this servers are brought close to the data generation point Known as the *Edge*. Edge computing can help to reduce latency in applications that require real-time or near real-time responses, such as autonomous vehicles, industrial automation, and healthcare applications. By processing data locally, edge computing can enable faster decision-making and reduce the risk of communication delays.

Edge computing can improve data privacy and security by enabling sensitive data to be processed locally at the edge, rather than being transmitted to a remote cloud server. This is particularly important for applications that deal with sensitive data, such as healthcare, finance, and government services.

Overall, edge computing is on the rise because it provides fast, efficient, and reliable computing resources at the edge of the network, enabling new applications and use cases that were not possible before. It can also help to reduce network congestion, improve network performance, and enhance data privacy and security.

2 Dataset 1: Traffic flow through a junction

2.1 Description

This dataset contains the number of cars passing through four junction measured at an hourly frequency. The measurements are taken over the course of nearly two years (from 2015-11-01 to 2017-06-30). The dataset is therefore comprised of:

- **Two** features : DateTime, Junction Number
- **One** output to be **regressed** : The traffic at a future time instant.
- **Total instances** : 48120

The problem requires us to estimate the quantity (number of cars passing through the junction) given its previous values. Hence the problem is called as an *auto-regression*. Auto-regression is used in many applications involving *time-series* data.

We have made use of data from junction 1 and in the year of 2015. This results in a dataset containing 1464 instances. This is done as the data between the junctions have no correlation to the other and the principles of time-series prediction can be illustrated with this small sample set.

Dataset Link: Traffic dataset

2.2 Algorithm

For reasons explained in Section. 2.5 we make use of the *Auto-Regressive Integrated Moving Average* model (ARIMA). ARIMA is a time series forecasting model that models the *autocorrelation* and *stationarity* in the data. The model consists of three components:

1. Autoregression (AR) component: This component models the relationship between the current value of the time series and its past values. It is based on the assumption that the current value of the time series depends on its past values.
2. Moving Average (MA) component: This component models the error term in the time series. It is based on the assumption that the current value of the time series depends on the past errors.
3. Integrated (I) component: This component models the *non-stationarity* in the time series. It involves *differencing* the time series to make it stationary.

The ARIMA model is specified using three parameters: p, d, and q. The parameter p is the order of the AR component, d is the degree of differencing required to make the time series stationary, and q is the order of the MA component.

2.3 Python Program

Import necessary libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from statsmodels.tsa.arima.model import ARIMA
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from sklearn.model_selection import train_test_split
from sklearn.model_selection import train_test_split
from statsmodels.tsa.stattools import adfuller
import numpy.fft as fft
```

Define functionality for ADF test

```
def adf_test(timeseries):
    adf_result = adfuller(timeseries)
    print(f'ADF Statistic: {adf_result[0]}')
    print(f'p-value: {adf_result[1]}')
    print('Critical Values:')
    for key, value in adf_result[4].items():
        print(f'   {key}: {value}')
    if adf_result[0] < adf_result[4]['5%']:
        print('Data is stationary')
    else:
        print('Data is non-stationary')
```

Get the data from 2015

```
data = pd.read_csv("traffic.csv")
data_2015 = data[pd.Series(map(lambda x:"2015" in x,data["DateTime"]))]
data_2015_junc1 = data_2015[data_2015["Junction"]==1]
```

Plot statistical features of the data

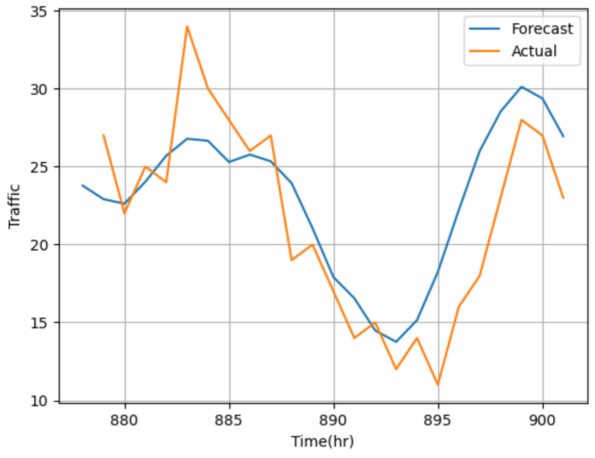
```
FFT = fft.fft(data_2015_junc1["Vehicles"])
plt.plot(abs(FFT)[0:len(FFT)//2])
plot_acf(data_2015_junc1["Vehicles"])
plot_pacf(data_2015_junc1["Vehicles"])
plt.show()
```

Defining the model, fitting it to training data and forecasting for the next 24 hours and plotting the error

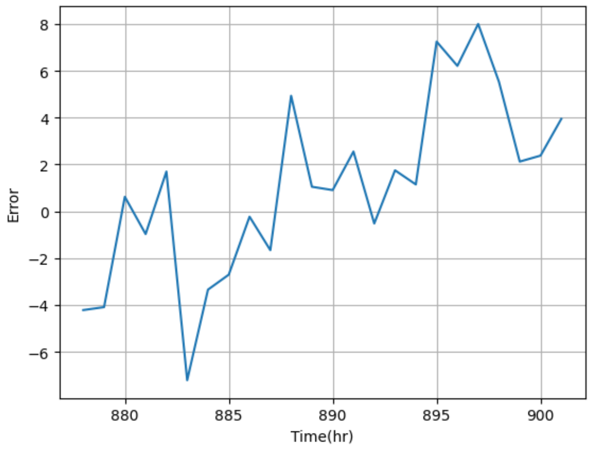
```
train_data, test_data = train_test_split(data_2015_junc1, test_size=0.4,
   shuffle=False)
model = ARIMA(train_data["Vehicles"], order=(30,0,1))
results = model.fit()
forecast = results.forecast(steps=24)
error = forecast - test_data["Vehicles"]
plt.plot(error)
np.sqrt((error**2).mean())
```

2.4 Experimental Results and Performance analysis

The model was trained with 878 samples, and predictions were made for the next 24 hours. The predictions, the difference(error) and the Mean-Squared Error (MSE) are shown below.



(a) Forecast and Actual



(b) Error

Figure 2: Forecast and Errors

The square root of the MSE is equal to 3.878. Which means that the forecast on average is off by about 4 cars.

2.5 Observation,Inferences and Conclusion

The ARIMA model is used on data which is *stationary*. A time-series is said to be stationary when its statistical properties i.e mean μ and variance σ^2 do not vary with time and the data must not have *seasonality* i.e there must be no repetition in the trends of the data.

The ARIMA model is controlled by 3 parameters named p,d,q as mentioned in Section. 2.2. The parameter d controls differencing in the model. This is used if the data appears to have certain trends in its mean μ. In our case, we can set d equal to zero as there is no such trend observed as shown in Fig. 3.

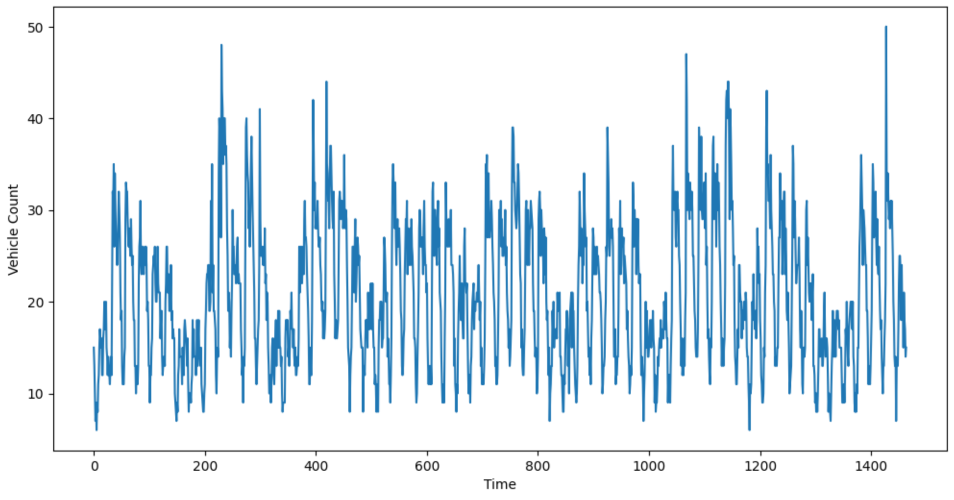
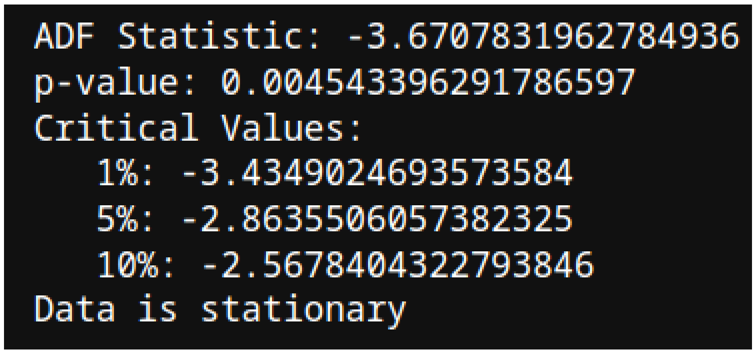


Figure 3: Traffic flow through junction 1 in 2015

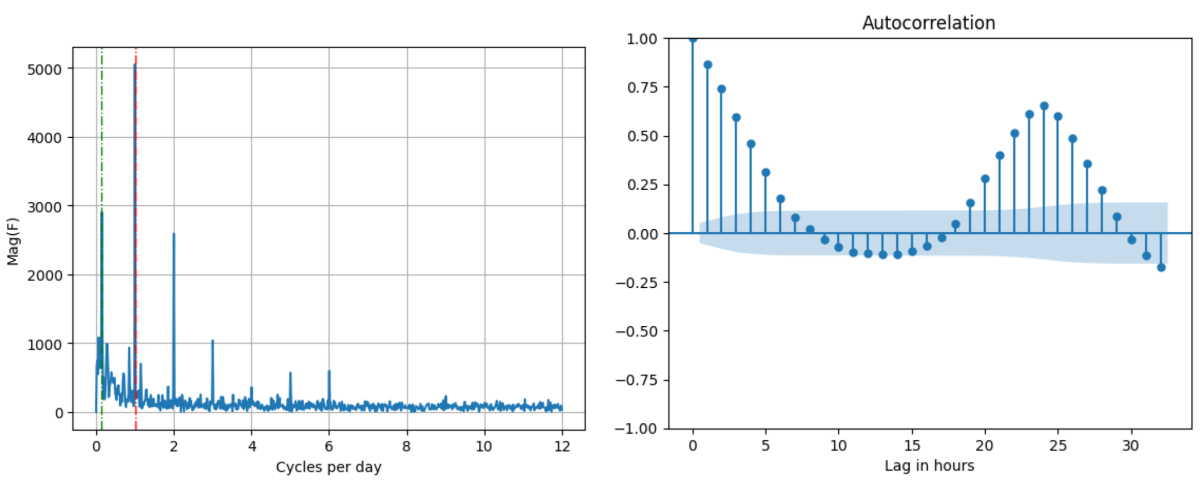
The *Augmented Dickey–Fuller test* (ADF-test) can determine if a given time-series data is stationary or not. Fig. 4 shows the results.



```
ADF Statistic: -3.6707831962784936
p-value: 0.004543396291786597
Critical Values:
   1%: -3.4349024693573584
   5%: -2.8635506057382325
   10%: -2.5678404322793846
Data is stationary
```

Figure 4: ADF test for data

However there is a clear seasonality component present in the data. This can be further proved by looking at the *Fourier transform* of the data and its *Auto-correlation* plots (see Fig. 5).



(a) Fourier Transform

(b) Auto-correlation

Figure 5: Periodic behaviour of data

From the auto-correlation plots, there is clearly a relation between the traffic observed at a time instant and its neighbouring instants. Also a correlation exists between the traffic at a lag of 24 hours, which is intuitive as we expect the traffic to roughly mimic its previous day. This is also observed in the fourier transform as well. An additional spike at $\frac{1}{7}$ cycles/day which is equal to 1 cycle per 7 days(or 1 week) is observed. This also makes sense as the traffic seen in one day of the week is likely to be seen again in the next week. From this we can conclude that there is seasonality in the data which was not revealed by the ADF-test. This has significant effect as ARIMA expects the data to lack seasonality. The predictions of the ARIMA model degrades as the predictions are extended over a larger period of time as shown by Fig. 6. Hence for such cases ARIMA must be used to forecast over shorter ranges of time. Another alternative is to use models which take seasonality into account. An example of such a model is the *Seasonal*-ARIMA (SARIMA).

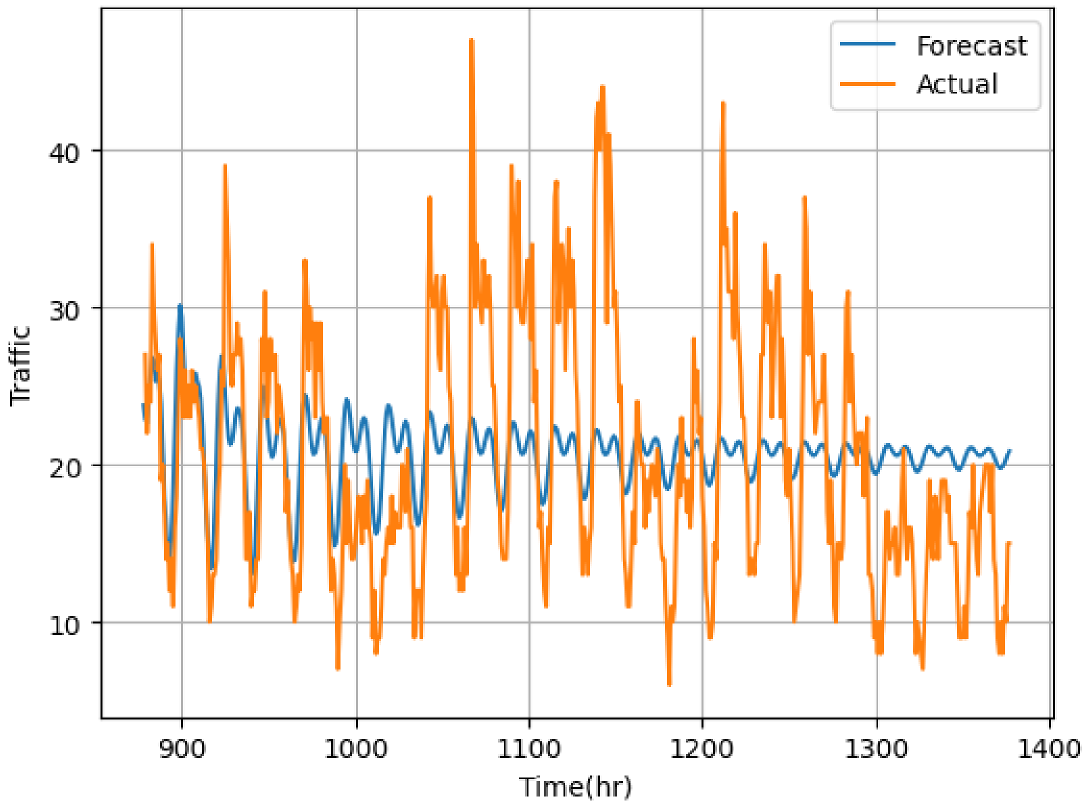


Figure 6: Forecast over a longer period of time

3 Dataset 2 : Compute Delay for a Edge Node

3.1 Description

This dataset contains execution times for offloaded image recognition tasks that were performed in a mobile edge computing environment. The tasks were offloaded from a client device (mobile edge node) to one of several edge servers, and the execution times were recorded for each server.
Detailed description :

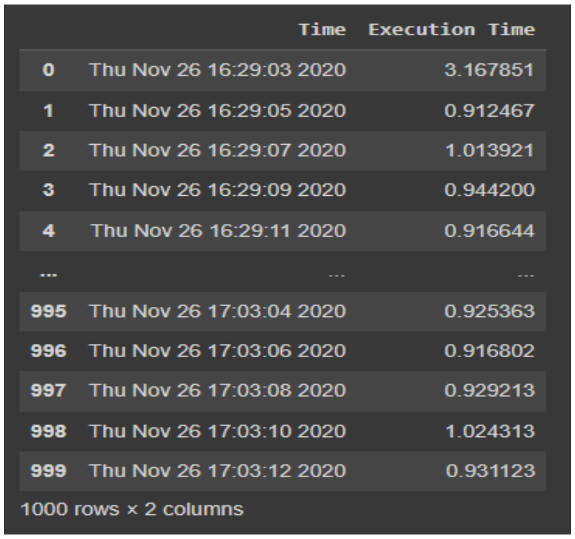
- **Number of Instances :** 4000
- **Number of Attributes :** 2 (Time: day, date, hours, minutes, second, year) & (Turnaround Task Execution time: in seconds)

There are four different edge servers in the dataset, each with its own processing power and memory capacity:

- MacBook Pro with a 1.4 GHz Quad-Core Intel Core i5 processor and 8 GB of LPDDR3 RAM
- MacBook Pro with a 2.5 GHz Dual-Core Intel Core i5 processor and 8 GB of DDR3 RAM
- Ubuntu virtual machine running on VirtualBox with 2 GB of RAM
- Raspberry Pi 4B with a quad-core 64-bit ARM Cortex-A72 CPU and 4 GB of RAM

The client device sends an image to one of these servers to be recognized, and the turnaround execution time is the time it takes for the server to receive the image, perform the recognition task, and send the result back to the client device. This time duration starts when the client device establishes a connection to the edge server and ends when the client device receives the recognition result.

In this study, we will be analyzing the execution times of offloaded image recognition tasks in a mobile edge computing environment.However, as we have observed significant variability in the execution times across different edge servers, We have chosen to focus exclusively on the data collected from the Raspberry Pi 4B server. To ensure the validity and reliability of our analysis, we will carefully filter and clean the data collected from the Raspberry Pi 4B server, and exclude any outlier or irrelevant observations. By focusing on a single server, we can provide a more in-depth understanding of the performance of edge computing for image recognition tasks.
Dataset Link: `Compute-Delay Dataset`



	Time	Execution Time
0	Thu Nov 26 16:29:03 2020	3.167851
1	Thu Nov 26 16:29:05 2020	0.912467
2	Thu Nov 26 16:29:07 2020	1.013921
3	Thu Nov 26 16:29:09 2020	0.944200
4	Thu Nov 26 16:29:11 2020	0.916644
...	...	...
995	Thu Nov 26 17:03:04 2020	0.925363
996	Thu Nov 26 17:03:06 2020	0.916802
997	Thu Nov 26 17:03:08 2020	0.929213
998	Thu Nov 26 17:03:10 2020	1.024313
999	Thu Nov 26 17:03:12 2020	0.931123

1000 rows × 2 columns

Figure 7: Data for Raspberry Pi 4B

3.2 Algorithm

For reasons explained in Section. 3.4, we will use a *Random Forest Regressor (RFR)* for our forecasting task. Unlike ARIMA, RFR is a machine learning algorithm that can handle both linear and nonlinear relationships between the features and the target variable. It works by constructing multiple decision trees and then aggregating their predictions to make the final prediction. The model is robust against overfitting and can handle high-dimensional datasets.

Random Forest Regressor is a supervised machine learning algorithm used for regression tasks. It is based on the concept of an ensemble of decision trees, where each tree is trained on a random subset of the data and a random subset of the features. During the training process, each tree makes a prediction for the target variable based on the input features, and the predictions from all trees are combined to generate the final prediction.

3.3 Python Program

Import necessary libraries

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from statsmodels.tsa.stattools import adfuller
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
```

Loading dataset and plotting

```
data = pd.read_csv("TATDescriptionDataset/RasberryPi.csv")
plt.figure(figsize=(12, 6))
plt.plot(data["Execution Time"])
plt.xlabel('Time')
plt.ylabel('Delay')
plt.show()
```

Plotting statistical features of the data

```
# Fourier Transform of the Data to visualize any periodicity
FFT = np.fft.fft(data["Execution Time"])
X = (abs(FFT))[0:len(FFT)//2]
X[0] = 0
plt.plot(X)
# Autocorrelation
plot_acf(data["Execution Time"])
plot_pacf(data["Execution Time"])
plt.show()
```

Defining functionality for ADF-test

```
def adf_test(timeseries):
    adf_result = adfuller(timeseries)
    print(f'ADF Statistic: {adf_result[0]}')
    print(f'p-value: {adf_result[1]}')
    print('Critical Values:')
    for key, value in adf_result[4].items():
        print(f'   {key}: {value}')
    if adf_result[0] < adf_result[4]['5%']:
        print('Data is stationary')
    else:
        print('Data is non-stationary')
adf_test(data['Execution Time'])
```

Defining model, random sampling of dataset for the random forest, training and evaluation of the model

```
train_data, test_data = train_test_split(data, test_size=0.2,shuffle=False)
lags = range(1, 11)
for lag in lags:
    train_data[f'lag_{lag}'] = train_data['Execution Time'].shift(lag)
    test_data[f'lag_{lag}'] = test_data['Execution Time'].shift(lag)

train_data = train_data.dropna()
test_data = test_data.dropna()
X_train = train_data.drop(['Time', 'Execution Time'], axis=1)
y_train = train_data['Execution Time']
X_test = test_data.drop(['Time', 'Execution Time'], axis=1)
y_test = test_data['Execution Time']

rf = RandomForestRegressor(n_estimators=100)
rf.fit(X_train, y_train)

y_pred = rf.predict(X_test)

mse = mean_squared_error(y_test, y_pred)
print(f'Mean squared error: {mse:.8f}')
```

3.4 Experimental Results and Performance analysis

The model is trained with 80% of the instances (about 800 samples) and tested with the remaining 200 samples. Number of *Trees* in the model had minimal effect on the result and hence kept at default of 100, depth of the tree is not defined in our model so that there is no *pruning*. The regression model evaluated has an MSE of 0.00485559. This indicates that the model's predictions are, on average, very close to the actual values in the test set.

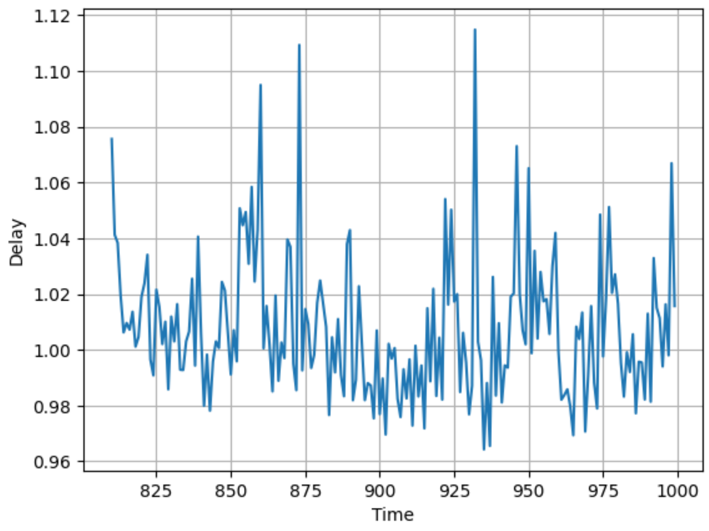


Figure 8: Delay estimates

3.5 Observations,Inferences and Conclusion

It can be observed that the data processes no periodicity and the same can be inferred from the Fast Fourier Transform (FFT) of the data Figure. 11a

As observed from Figure. 11b, Data exhibits low or no autocorrelation, it suggests that the series is more random and less predictable, and that the current value of the series is not strongly influenced by its past values.

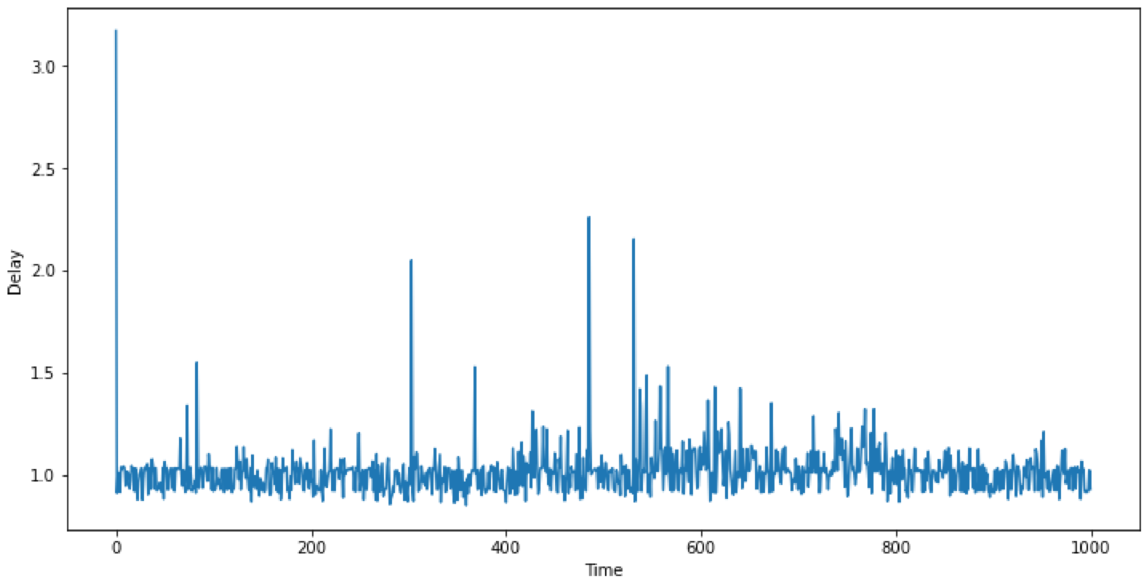
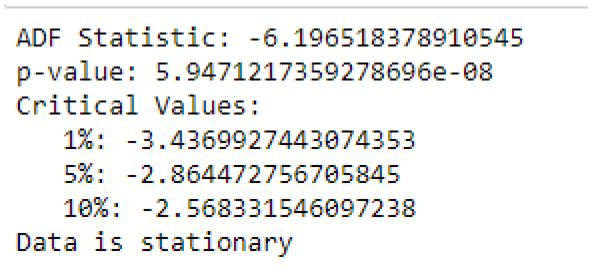


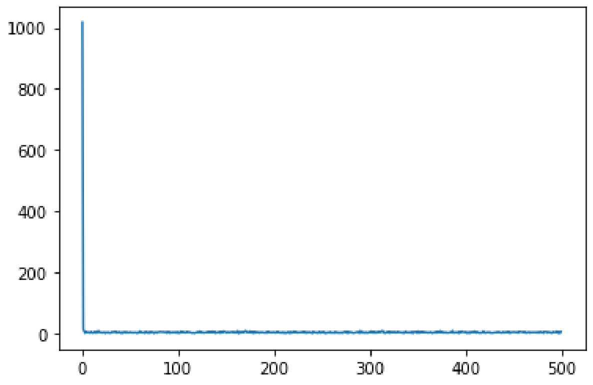
Figure 9: Delay Vs Time of Data from Raspberry Pi 4B

The *Augmented Dickey–Fuller test* (ADF-test) can determine if a given time-series data is stationary or not. Fig. 10 shows the results.

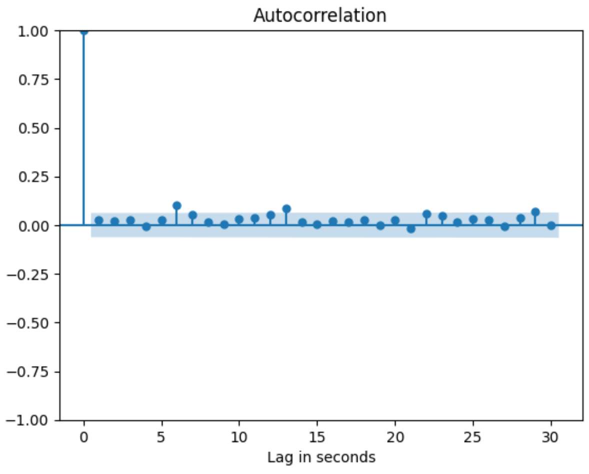


```
ADF Statistic: -6.196518378910545
p-value: 5.9471217359278696e-08
Critical Values:
   1%: -3.4369927443074353
   5%: -2.864472756705845
   10%: -2.568331546097238
Data is stationary
```

Figure 10: ADF test for data



(a) Fourier Transform



(b) Auto-correlation

Figure 11: Fast Fourier Transform and Auto-Correlation for analysis of Periodic behaviour of data

4 Dataset 3 : Metro Interstate Traffic Volume Data Set

4.1 Description

This Metro Interstate Traffic Volume dataset contains hourly traffic volume data for the I-94 Interstate highway in the Minneapolis-St. Paul metropolitan area of Minnesota, USA. The dataset was collected by the Minnesota Department of Transportation from 2012 to 2018, and includes 48,204 observations.
Detailed description :

- **Number of Instances :** 48204
- **Number of Attributes :** 13

Each observation in the dataset consists of 13 attributes, including a Boolean indicator for whether the observation was made during a holiday, various weather-related features such as temperature and precipitation, and the hourly traffic volume in both directions on the I-94. The dataset also includes the date and time of each observation, as well as indicators for the year, month, day, and hour of the observation.

The goal of this dataset is to predict the hourly traffic volume on the I-94 based on the various features included in the dataset, which could be useful for transportation planning and management. The dataset has been used in various studies and competitions for time series forecasting and regression analysis. The dataset is obtained from measurements taken over the course of 6 years from 2012 to 2018. Once again similar to Dataset 1 we will consider the data points from the year 2012 only. This yields us a total of 2559 samples.
Dataset Link: `Metrostate-traffic Dataset`

4.2 Algorithm

Due to the availability of extra features such as weather and temperature, we have chosen to use a more sophisticated model for this dataset. The algorithm used for this dataset is the *Recurrent neural network* (RNN). In particular the Recurrent units are *Long Short Term Memory* (LSTM) units. An LSTM unit is a type of recurrent *neuron* which also maintains a type of *state*. The state here is the Long term memory and a short term memory. Given an input to the LSTM unit, its values of long term and short term memory are updated. LSTM(s) are very popular in forecasting problems due to their ability to form a kind of memory. When given a series of past inputs, it can predict the future outcome by using the sequence of data points, iteratively updating its internal state and hence generate the new prediction.

4.3 Python Program

Import necessary libraries

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error
```

Loading dataset, filtering out points from 2012

```python
df = pd.read_csv('Metro_Interstate_Traffic_Volume.csv')
df = df[pd.Series(map(lambda x:"2012" in x,df["date_time"]))]
```

Scale the features and targets to a value between 0 and 1

```python
scaler = MinMaxScaler()
df[['temp', 'rain_1h', 'snow_1h', 'clouds_all', 'traffic_volume']] = scaler
    .fit_transform(df[['temp', 'rain_1h', 'snow_1h', 'clouds_all', '
    traffic_volume']])
```

Plotting statistical features of the data

```
FFT = np.fft.fft(data_2012["traffic_volume"])
#FFT = fft.fftshift(FFT)
X = np.linspace(0,24/2,len(FFT)//2)
F = abs(FFT)[0:len(FFT)//2]
F[0] = 0
plt.plot(X,F)
plt.xlabel("Cycles per day")
plt.ylabel("Mag(F)")
plt.axvline(x=1, color='red', linestyle='-.',linewidth=1)
plt.axvline(x=1/7, color='green', linestyle='-.',linewidth=1)
plt.grid()
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
plot_acf(data_2012["traffic_volume"])
plt.xlabel("Lag in hours")
plot_pacf(data_2012["traffic_volume"])
plt.xlabel("Lag in hours")
plt.show()
```

Define functionality for ADF-test

```
from statsmodels.tsa.stattools import adfuller

def adf_test(timeseries):
    adf_result = adfuller(timeseries)
    print(f'ADF Statistic: {adf_result[0]}')
    print(f'p-value: {adf_result[1]}')
    print('Critical Values:')
    for key, value in adf_result[4].items():
        print(f'   {key}: {value}')
    if adf_result[0] < adf_result[4]['5%']:
        print('Data is stationary')
    else:
        print('Data is non-stationary')
adf_test(data_2012["traffic_volume"])
```

Generate lagged time series. This will be used for training

```
lags = range(1, 25)
for lag in lags:
    df[f't-{lag}'] = df['traffic_volume'].shift(lag)
```

Dropping unnecessary textual features

```
df.drop("holiday",axis=1,inplace=True)
df.drop("weather_main",axis=1,inplace = True)
df.drop("weather_description",axis=1,inplace= True)
```

Defining model, training and testing

```
# select features and target variable
X_train =train_data.drop(['date_time', 'traffic_volume'], axis=1).values
y_train =train_data['traffic_volume'].values
X_test = test_data.drop(['date_time', 'traffic_volume'], axis=1).values
y_test = test_data['traffic_volume'].values
# reshape the input data to a 3D array for LSTM network
X_train = np.reshape(X_train, (X_train.shape[0], 1, X_train.shape[1]))
```

```
X_test = np.reshape(X_test, (X_test.shape[0], 1, X_test.shape[1]))
# Create and compile LSTM model
model = tf.keras.models.Sequential([
    tf.keras.layers.LSTM(units=64, input_shape=(X_train.shape[1], X_train.
  shape[2])),
    tf.keras.layers.Dense(units=1)
])

model.compile(loss='mean_squared_error', optimizer=tf.keras.optimizers.Adam
  ())
# Convert input data to TensorFlow tensor objects
X_train_tf = tf.convert_to_tensor(X_train)
y_train_tf = tf.convert_to_tensor(y_train)
# Fit the LSTM model
history = model.fit(X_train_tf, y_train_tf, epochs=50, batch_size=72,
  validation_split=0.1, verbose=1, shuffle=False)
# Evaluate the model on test data
X_test_tf = tf.convert_to_tensor(X_test)
y_test_tf = tf.convert_to_tensor(y_test)

test_loss = model.evaluate(X_test_tf, y_test_tf)

print('Test Loss:', test_loss)
plt.plot(y_pred,label="Forecast")
plt.plot(y_test,label = "Actual")
plt.legend()
plt.xlabel("Time")
plt.ylabel("Normed traffic volume")
```

4.4 Experimental Results and Performance analysis

The network is defined with the following hyper-parameters:

1. 64 LSTM units.
2. One dense layer with a single unit to aggregate the results from the LSTM units.

The *Adaptive moment estimation* (ADAM) optimizer is used for the updation of the weights and biases of the network. 1559 samples are used for training. The training uses *Stocastic gradient descent* also called as mini-batched training. The mini-batch size is 72 samples. Also 10% of the training samples are held for validation. The model is trained for 50 epochs.

The MSE reported for the test dataset is 0.00569. The model takes lagged samples from the series (12 samples) and makes a new prediction. The predictions are shown in Fig. 12.

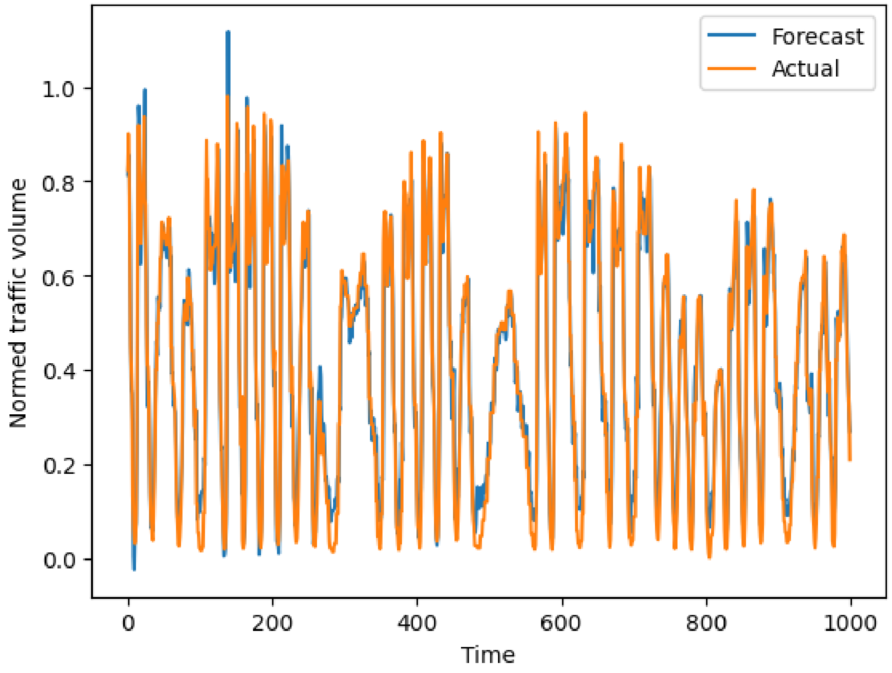


Figure 12: Forecast vs Actual from LSTM

4.5 Observations,Inferences and Conclusions

This dataset is quite similar to Dataset-1, and hence we performed similar analysis on the data. The ADF test (Fig. 14) reports that the data is stationary. This seems true as the mean and variance are constant and there appears to be no seasonality as seen in Fig. 13.

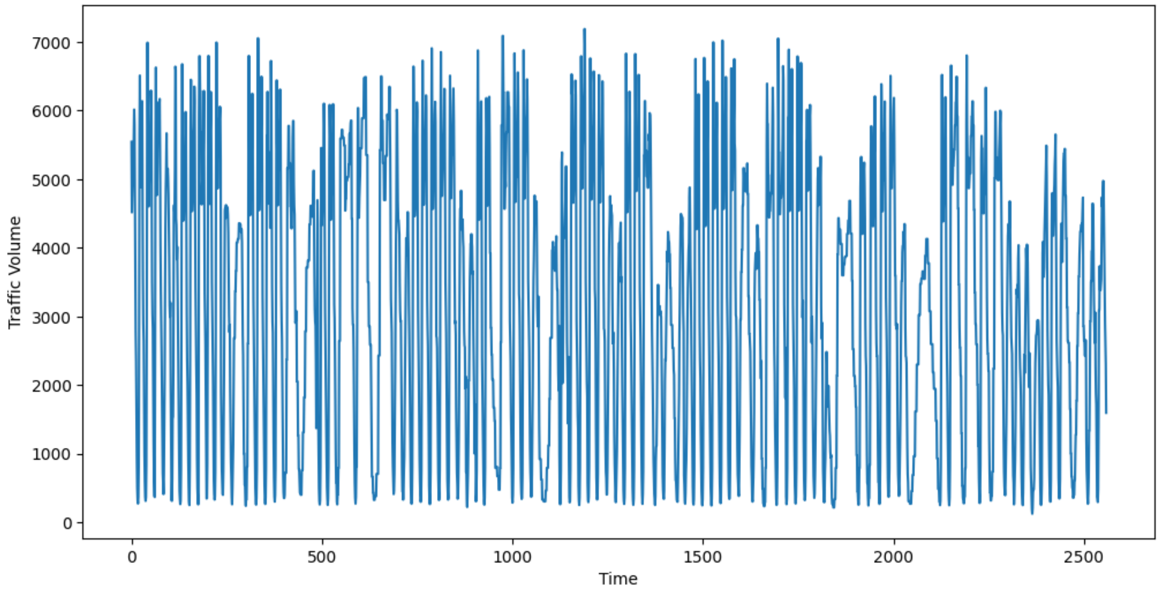
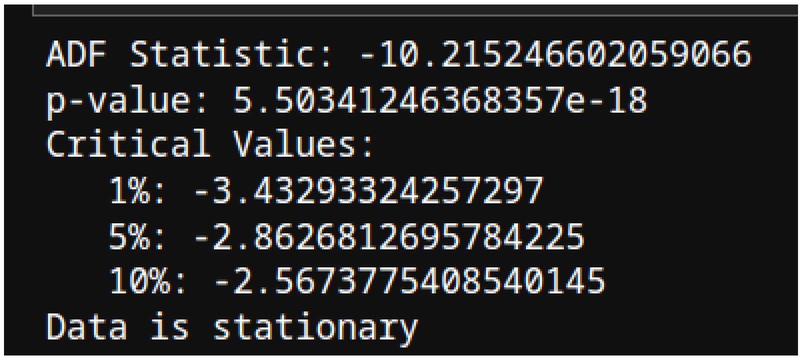


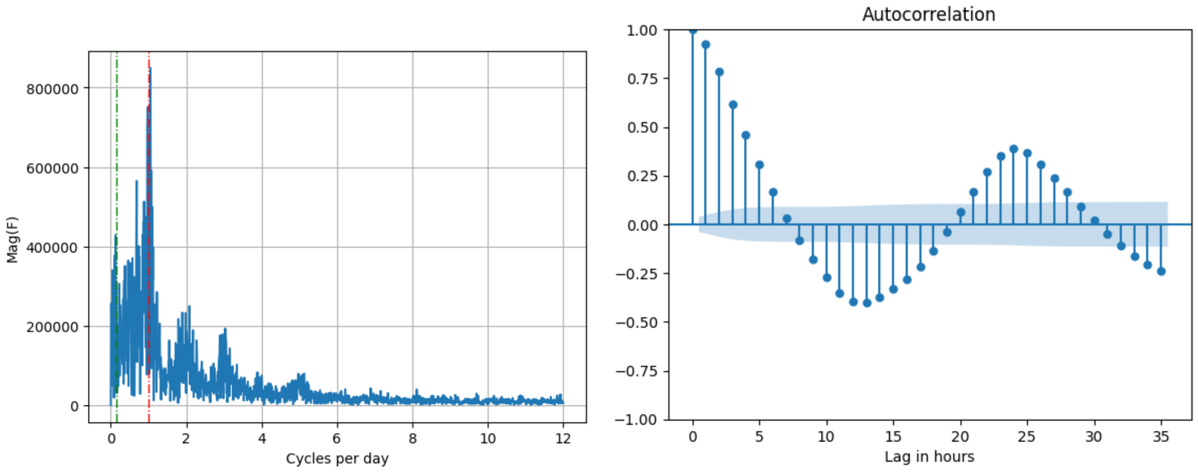
Figure 13: Metro-state traffic from 2012



```
ADF Statistic: -10.215246602059066
p-value: 5.50341246368357e-18
Critical Values:
   1%: -3.43293324257297
   5%: -2.8626812695784225
   10%: -2.5673775408540145
Data is stationary
```

Figure 14: ADF test for dataset

However we once again notice seasonal components revealed by the Fourier transfrom and the Auto-correlation function (see Fig. 15).



(a) Fourier Transform (b) Auto-correlation

Figure 15: Periodic behaviour of data

There also be some additional harmonics involved in the Fourier transform. Due to this observation ,the poor performance of ARIMA on longer periods of time for seasonal data and the availabilty of additional features, we made use of a Recurrent neural network for this dataset.